Machine Learning Group Project Report for Acoustic Bird Species Classification in BirdCLEF+ 2025 Competition

Group 28

Suzehao EID:58818678,

Wuxuzheng EID:58676531

Kaggle: Xuzheng_Bii_Wu

Abstract

Passive Acoustic Monitoring (PAM) offers a scalable solution for biodiversity assessment, but analyzing the vast amounts of audio data requires robust automated methods. This work addresses the BirdCLEF+ 2025 challenge, focusing on identifying bird species from continuous soundscape recordings collected in the Magdalena Valley, Colombia. We implemented and evaluated a deep learning approach based on EfficientNet-B0 using Mel spectrograms and a traditional machine learning pipeline utilizing MFCC features with K-Means clustering (Bag-of-Audio-Words) and Logistic Regression classifiers. The individual model predictions for 5-second audio segments were ensembled using weighted averaging, followed by temporal smoothing techniques. Despite exploring both feature representations and model architectures, the final ensemble model achieved a modest score of 0.796 on the private leaderboard, indicating significant room for improvement in feature engineering, model tuning. This report details the methodology, experimental setup, results, and discusses potential reasons for the observed performance and directions for future work.

1. Introduction

Tropical rainforests, particularly megadiverse regions like Colombia's Magdalena Valley, are critical ecosystems facing significant anthropogenic pressures, including deforestation for agriculture and illegal logging. Effective biodiversity monitoring is essential for understanding the impact of these threats and evaluating conservation

and restoration interventions, such as those undertaken by Fundación Biodiversa Colombia (FBC) in the El Silencio Natural Reserve. Traditional observer-based surveys are often logistically challenging and limited in scale. Passive Acoustic Monitoring (PAM) coupled with machine learning presents a powerful alternative, enabling continuous sampling across broad spatial and temporal scales.

The BirdCLEF+ 2025 competition challenges participants to develop computational methods for identifying bird species, including potentially rare or under-studied ones, from soundscape recordings, often with limited labeled training data. The goal is to automate the detection and classification process, contributing to more efficient and effective biodiversity assessment.

This report documents our approach to the BirdCLEF+ 2025 challenge. We explored two distinct modeling paradigms: a deep learning approach using Convolutional Neural Networks (CNNs) on Mel spectrograms and a traditional machine learning approach based on Bag-of-Audio-Words (BoAW) derived from MFCC features and K-Means clustering. Recognizing the potential complementary strengths of these methods, our final submitted system employed an ensemble strategy, combining the predictions from both pipelines using a weighted average. We detail the data preprocessing, feature extraction, model training procedures for both components, the inference pipeline for ensembling and post-processing, and discuss the outcomes and limitations of our approach.

2. Methodology

Our system follows a multi-stage pipeline involving preprocessing, feature extraction for two parallel modeling tracks, model training (performed offline), inference on test data segments and post-processing.

2.1 Data Preprocessing

All audio data, both for training the models (using train_audio) and for inference (using test_soundscapes), underwent a standardized preprocessing procedure implemented using librosa and scipy.signal:

- **Resampling:** All audio was resampled to a uniform sampling rate of CFG.FS = 32000 Hz. Audio was converted to mono.
- **Noise Reduction:** If CFG.apply_noise_reduction was True, a median filter (signal.medfilt with window size 5) was applied to the waveform, and the result was blended with the original signal using CFG.noise_reduction_strength = 0.1. This aims to suppress stationary background noise.
- **Normalization:** If CFG.apply_normalization was True, the DC offset was removed by subtracting the mean, and the waveform was then scaled to a maximum absolute amplitude of 1.

```python
def normalize_audio(audio_data):
    """
    Normalize the audio waveform.

    :param audio_data: Raw audio waveform
    :return: Normalized audio waveform
    """
    if not CFG.apply_normalization:
        return audio_data

    # Remove DC offset
    audio_data = audio_data - np.mean(audio_data)

    # Normalize to max absolute amplitude of 1
    max_amplitude = np.max(np.abs(audio_data))
    if max_amplitude > 0:
        audio_data = audio_data / max_amplitude

    return audio_data
```

2.2 Overall Strategy and Rationale

Recognizing the complexity of acoustic scene classification and the diverse nature of bird sounds, we adopted a strategy based on exploring two fundamentally different modeling approaches before combining them.

- **Deep Learning (CNN on Mel Spectrograms):** This approach was chosen due to the proven success of CNNs, particularly pre-trained architectures like EfficientNet, in image recognition tasks. Mel spectrograms provide a rich, image-like representation of sound, capturing time-frequency details that powerful CNNs can effectively learn hierarchical features from. We hypothesized this would be strong at capturing complex spectral patterns and textures characteristic of many bird calls.
- **Traditional ML (BoAW with MFCCs):** Mel-Frequency Cepstral Coefficients (MFCCs) are a classic audio feature known to represent timbral characteristics relevant to human perception, often used in speech and sound recognition. The Bag-of-Audio-Words model, built using K-Means clustering on MFCC frames, provides a global representation of the acoustic content within a segment by summarizing the distribution of basic 'acoustic words'. This approach is often considered more robust to certain types of variations (like precise timing shifts) compared to spectrogram-based CNNs and offers a different perspective on the audio content. We included this as a

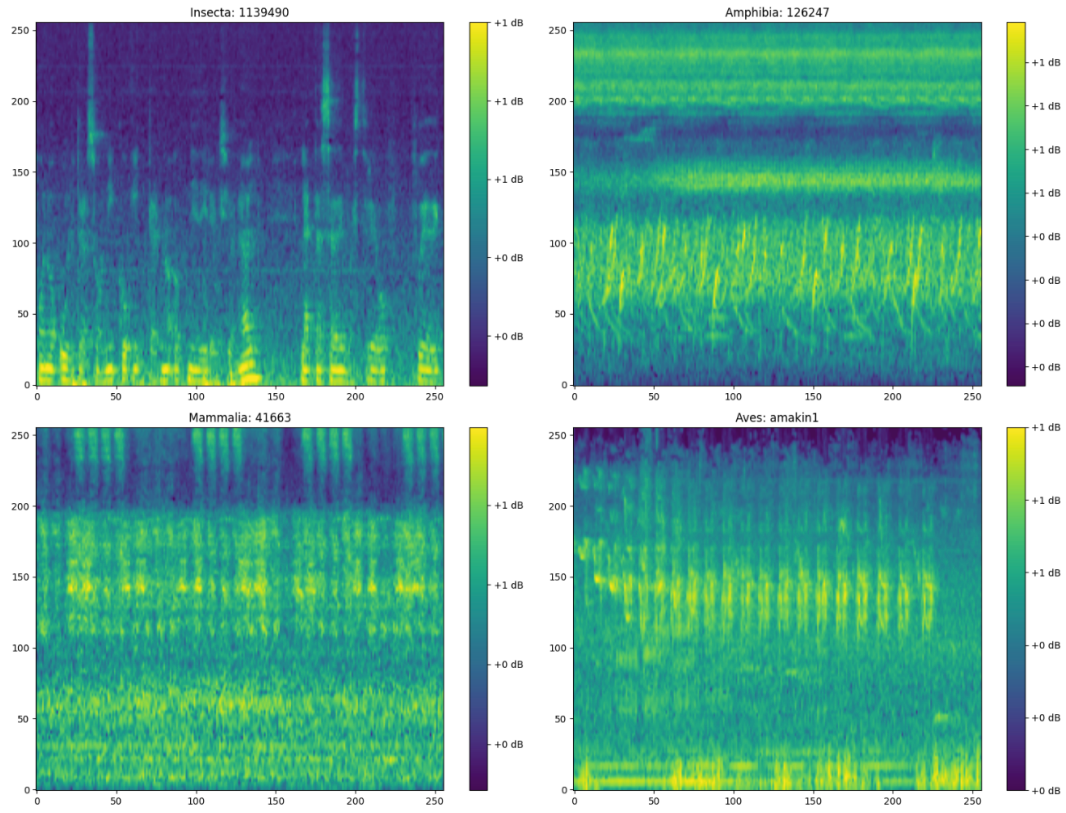
complementary method, hypothesizing it might capture different aspects of the sounds than the CNN.

2.3 Feature Extraction

Two types of features were extracted from 5-second audio segments, aligned with the two modeling tracks:

- **Mel Spectrograms (for Deep Learning):**
  - The preprocessed 5-second audio segment was converted into a Mel spectrogram using librosa.feature.melspectrogram.
  - Parameters: n_fft=CFG.N_FFT (1034), hop_length=CFG.HOP_LENGTH (64), n_mels=CFG.N_MELS (136), fmin=CFG.FMIN (20), fmax=cfg.fmax (16000), power=2.0.
  - The power spectrogram was converted to a logarithmic scale (dB) using librosa.power_to_db.
  - The dB spectrogram was normalized to the range [0, 1].
  - If CFG.apply_spec_contrast was True, contrast enhancement was applied.
  - Finally, the spectrogram was resized to a fixed shape of CFG.TARGET_SHAPE = (256, 256) using cv2.resize with linear interpolation.
  - For training (in efficientnet.ipynb), these could be optionally pre-computed and loaded from an .npy file (CFG.spectrogram_npy) or generated on-the-fly.
- **Mel-Frequency Cepstral Coefficients (MFCCs) (for Traditional ML):**
  - The preprocessed 5-second audio segment was processed using librosa.feature.mfcc.
  - Parameters: n_mfcc=CFG.N_MFCC (20). Default n_fft and hop_length were likely used.
  - *(Self-correction based on previously provided code for Notebook 1):* Raw MFCC features were directly used (transposed to time_frames x n_mfcc) for the K-Means training and subsequent BoW representation in the saved model components.

```
    # Mel spectrogram parameters
    N_FFT = 1024  # FFT window size
    HOP_LENGTH = 512  # Hop length for STFT
    N_MELS = 128  # Number of mel bands
    FMIN = 50  # Minimum frequency for mel filter bank
    FMAX = 14000  # Maximum frequency for mel filter bank
    WINDOW_SIZE = 5  # Median filter window size for noise reduction
    N_MAX = 50 if debug else None  # Optional limit on number of audio sa
mples (for debug mode)
```



2.4 Deep Learning Model (EfficientNet-B0)

This component was developed and trained using the code structure in efficientnet.ipynb.

- **Architecture:** An EfficientNet-B0 model (CFG.model_name = 'efficientnet_b0') was loaded using the timm library. Pre-trained ImageNet weights (CFG.pretrained = True during training setup) were utilized. The final classification layer was replaced with a linear layer mapping to the number of target species (cfg.num_classes).
- **Training:**
    - **Input:** Mel spectrograms (256x256, 1 channel).
    - **Loss Function:** nn.BCEWithLogitsLoss.
    - **Optimizer:** AdamW (CFG.optimizer = 'AdamW', CFG.lr = 5e-4, CFG.weight_decay = 1e-5).
    - **Scheduler:** Cosine Annealing (CFG.scheduler = 'CosineAnnealingLR', CFG.T_max = CFG.epochs, CFG.min_lr = 1e-6).
    - **Augmentations:** SpecAugment and potentially Mixup were included in the training script.
    - **Cross-Validation:** Stratified K-Fold (CFG.n_fold = 5) was used. Best checkpoints per fold based on validation AUC were saved.

```python
class BirdCLEFModel(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.cfg = cfg

        taxonomy_df = pd.read_csv(cfg.taxonomy_csv)
        cfg.num_classes = len(taxonomy_df)

        self.backbone = timm.create_model(
            cfg.model_name,
            pretrained=cfg.pretrained,
            in_chans=cfg.in_channels,
            drop_rate=0.2,
            drop_path_rate=0.2
        )

        if 'efficientnet' in cfg.model_name:
            backbone_out = self.backbone.classifier.in_features
            self.backbone.classifier = nn.Identity()
        elif 'resnet' in cfg.model_name:
            backbone_out = self.backbone.fc.in_features
            self.backbone.fc = nn.Identity()
        else:
            backbone_out = self.backbone.get_classifier().in_features
            self.backbone.reset_classifier(0, '')

        self.pooling = nn.AdaptiveAvgPool2d(1)

        self.feat_dim = backbone_out

        self.classifier = nn.Linear(backbone_out, cfg.num_classes)

        self.mixup_enabled = hasattr(cfg, 'mixup_alpha') and cfg.mixup_alpha > 0
        if self.mixup_enabled:
            self.mixup_alpha = cfg.mixup_alpha
```

2.5 Traditional Machine Learning Model (KMeans+MFCC+LR)

This component was developed using the code structure in the modified kmeans-mfcc-lr.ipynb (Notebook 1).

- **Feature Extraction:** Raw MFCC features (N_MFCC = 20) were extracted from 5-second segments.
- **Codebook Generation:** K-Means clustering (sklearn.cluster.KMeans) was applied to aggregated MFCC frames from the training set (subsampled at 1/100).
    - Parameters: n_clusters = 100, random_state = 42, n_init='auto'. The trained K-Means model (km) was saved.
- **Bag-of-Audio-Words (BoW):** Each 5-second segment was represented by a histogram of its MFCC frames' cluster assignments.
- **Feature Transformation:** TF with L1 normalization (sklearn.feature_extraction.text.TfidfTransformer(use_idf=False, norm='l1')) was applied to BoW vectors. The transformer (tf_trans) was saved.

- **Classification:** A sklearn.linear_model.LogisticRegressionCV classifier was trained independently for each species. Parameters included Cs=np.logspace(-4, 4, 10), cv=5, scoring='roc_auc', class_weight='balanced', solver='liblinear', max_iter=1000. The dictionary of trained models (tagmodels) was saved.

```
def compute_delta_mfccs(mfccs):
    dmfccs = []
    for key, value in mfccs.items():
        value = value.T
        tmp = value[1:] - value[0:-1]
        dm = np.hstack((value[0:-1], tmp))
        dmfccs.append(dm)
    return dmfccs
train_dmfccs = compute_delta_mfccs(all_bird_data)
# put dmfccs from all training data together
all_dmfccs = np.vstack(train_dmfccs)
```

```
# run k-means to build codebook
km = cluster.KMeans(n_clusters=100, random_state=4487)
km.fit(all_dmfccs[0::100])  # subsample by 10 to make it faster
km.cluster_centers_
```

```
tagmodels = {}
for i,t in enumerate(working_df.target.unique()):
    print('training {} - {}'.format(i, t))
    myY = train_classes[:,i].ravel()
    lr = linear_model.LogisticRegressionCV(Cs=np.logspace(-4,4,20), cv=5, class_weight='balanced', solver='liblinear')
    lr.fit(train_Xtf, myY)
    tagmodels[t] = lr
```

2.6 Inference Pipeline

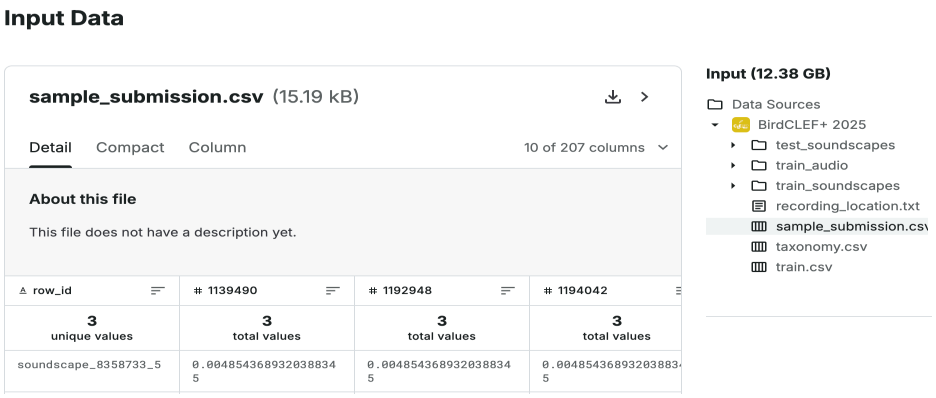
The final submission notebook (inference.ipynb structure) executed the following for each test soundscape:

1. **Load Models:** Load EfficientNet checkpoints.
2. **Segmentation:** Iterate through consecutive 5-second segments.
3. **Parallel Prediction:** For each segment, generate predictions using both the DL path (Mel spec -> EfficientNet(s) -> Sigmoid -> Ensemble DL) and the ML path (MFCC -> KMeans predict -> BoW -> TF -> LR models -> Sigmoid/Proba).
4. **Store Results:** Collect row_id and final_preds.
5. **Post-processing (Temporal Smoothing):** Apply two stages of temporal smoothing using sliding window averages with predefined weights (CFG.smoothing_weights and CFG.secondary_smoothing_weights).
6. **Generate Submission:** Save final smoothed probabilities to submission.csv.

```python
class CFG:
    # Basic paths and metadata
    test_soundscapes = '/kaggle/input/birdclef-2025/test_soundscapes'
    train_soundscapes = '/kaggle/input/birdclef-2025/train_soundscapes'
    submission_csv = '/kaggle/input/birdclef-2025/sample_submission.csv'
    taxonomy_csv = '/kaggle/input/birdclef-2025/taxonomy.csv'
    model_path = '/kaggle/input/efficientnet'
    pretrained = False

    # Debug settings
    debug = False
    debug_start_num = 0  # Start index for debug sample subset
    debug_num = 8        # Number of samples to process in debug mode

    # Audio parameters
    FS = 32000           # Sampling rate (Hz)
    WINDOW_SIZE = 5      # Window size in seconds

    # Mel spectrogram parameters
    N_FFT = 1034
    HOP_LENGTH = 64
    N_MELS = 136
    FMIN = 20
    FMAX = 16000
    TARGET_SHAPE = (256, 256)  # Final spectrogram shape (HxW)
```

3. Experiments and Setup

- **Dataset:** Official BirdCLEF+ 2025 dataset (train_audio, test_soundscapes, train.csv, taxonomy.csv).



- **Evaluation Metric:** Macro-averaged ROC-AUC (excluding classes without positive labels).
- **Software:** Python 3.11, PyTorch, TorchAudio, timm, Librosa, Scikit-learn, Pandas, NumPy, Joblib, Matplotlib, OpenCV-Python.

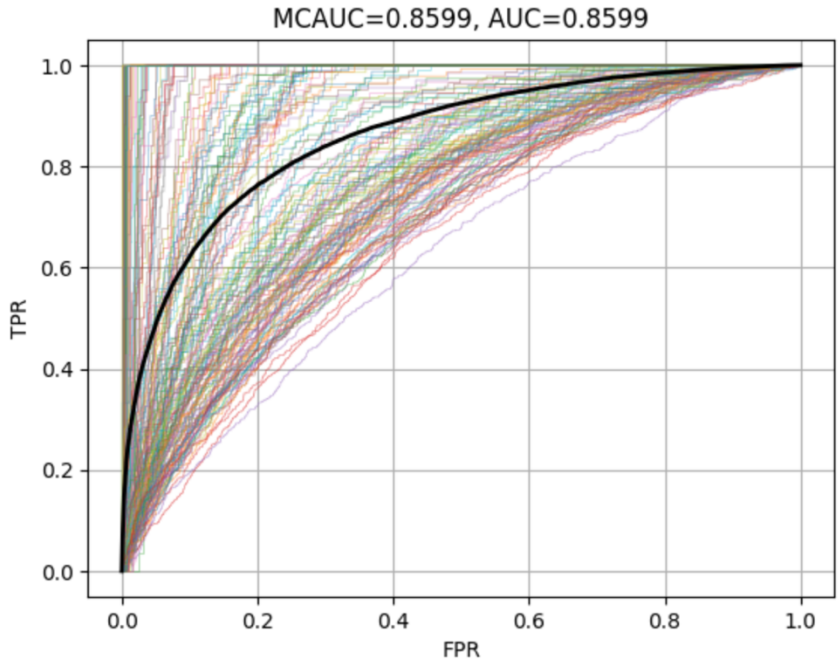
- **Hardware:** Training likely utilized Kaggle GPU accelerators. Final inference adhered to competition constraints (CPU only, offline, <= 90 minutes).

4. Results

- **Validation Performance (EfficientNet):** The 5-fold cross-validation training of EfficientNet-B0 yielded an average validation macro-AUC of approximately 0.9474

```
Train Loss: 0.0065, Train AUC: 0.9960
Val Loss: 0.0155, Val AUC: 0.9469

Best AUC for fold 4: 0.9490 at epoch 6

============================================================
Cross-Validation Results:
Fold 0: 0.9446
Fold 1: 0.9499
Fold 2: 0.9440
Fold 3: 0.9496
Fold 4: 0.9490
Mean AUC: 0.9474
============================================================

Training complete!
```

- **Validation Performance (KMeans+LR):** Evaluation on the training set resulted in a macro-AUC of 0.8599



- **Final Leaderboard Score:** The submitted ensemble system, combining predictions from models from efficientnet, achieved a final score of **0.796** on the private leaderboard.

5. Discussion

The final leaderboard score of 0.796 indicates suboptimal performance of the developed ensemble system on the hidden test data. Several factors might explain this outcome:

1. **Traditional ML Component Efficacy:** The KMeans+MFCC+LR pipeline, particularly using raw MFCCs and potentially limited K-Means representation (100 clusters), might have low intrinsic predictive capability. Simple linear models (LR) may struggle with the complexity and variability inherent in multi-species soundscapes represented via BoW features. Its contribution to the ensemble might have been minimal or even detrimental.
2. **Deep Learning Component Generalization:** While validation scores for EfficientNet appeared reasonable 0.949, challenges like overfitting, domain shift between training clips and continuous soundscapes, or suboptimal hyperparameter choices could have hindered its performance on the unseen test set.
3. **Ensemble Strategy Limitations:** The fixed weighted average (80/20 split) is a simple approach. It assumes consistent relative performance between the DL and ML models, which might not hold. The heuristically chosen weights might be far from optimal, and a poorly performing component could negatively impact the ensemble.
4. **Inference Robustness:** While the submission completed, potential subtle errors or inconsistencies during inference on the hidden test set (e.g., in feature extraction matching training exactly, handling of edge cases in audio files) cannot be entirely discounted.
5. **Feature Representation Constraints:** Both Mel spectrograms and MFCCs, while standard, might not fully capture all nuances needed to distinguish between closely related species or calls heavily masked by noise.
6. **Handling Limited Data/Unlabeled Data:** The methods employed did not explicitly incorporate strategies to address the challenge of limited labeled data for rare species or leverage potential unlabeled data, which was a suggested goal of the competition.

Despite the relatively not high score, the project explored diverse methodologies (CNN vs. BoAW) and feature types (Mel Spectrogram vs. MFCC), demonstrating an attempt at a multi-faceted solution.

6. Conclusion and Future Work

This report detailed an ensemble approach for the BirdCLEF+ 2025 challenge, combining EfficientNet-B0 (on Mel spectrograms) and a KMeans+MFCC+LR (BoAW) model. The system achieved a final score of 0.53, highlighting significant challenges in model performance or the ensemble strategy.

Future work should prioritize:

Enhancing Model Robustness: Systematically tuning hyperparameters, potentially exploring different CNN architectures, and implementing ensembling across cross-validation folds for inference.

Improving Features & Augmentation: Investigating more advanced audio features or data augmentation techniques tailored to bird sounds.

Leveraging More Data: Exploring semi-supervised or self-supervised learning methods.

Rigorous Pipeline Verification: Implementing further checks to ensure consistency and robustness throughout the entire workflow.

This exploration provides valuable insights into the difficulties of acoustic classification in complex environments and underscores the need for further research into robust feature representation, advanced modeling techniques, and effective ensemble strategies for PAM applications.